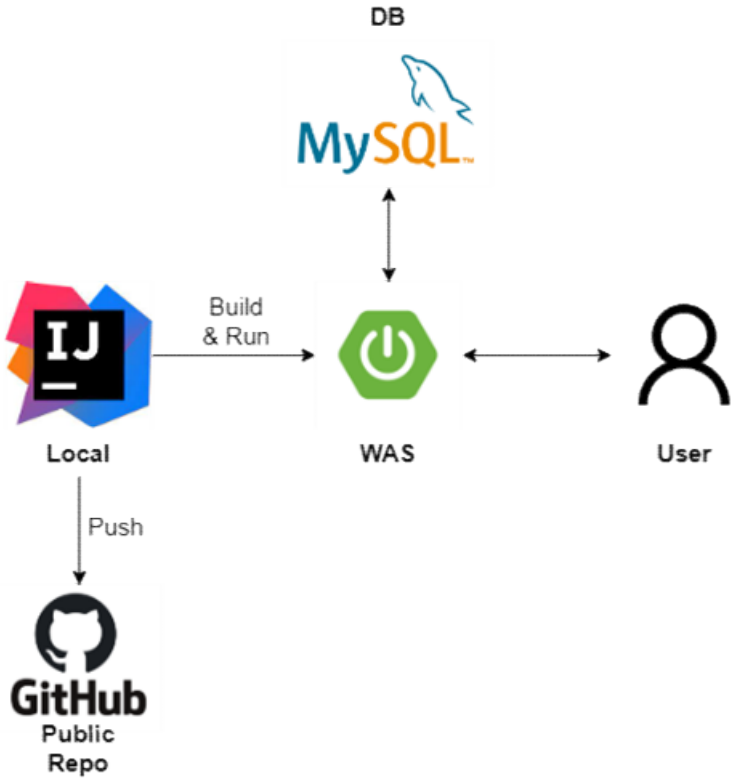
SUPER SPACE

- Application Modernization Process -

Author: Y.Hun

0. Basic Step



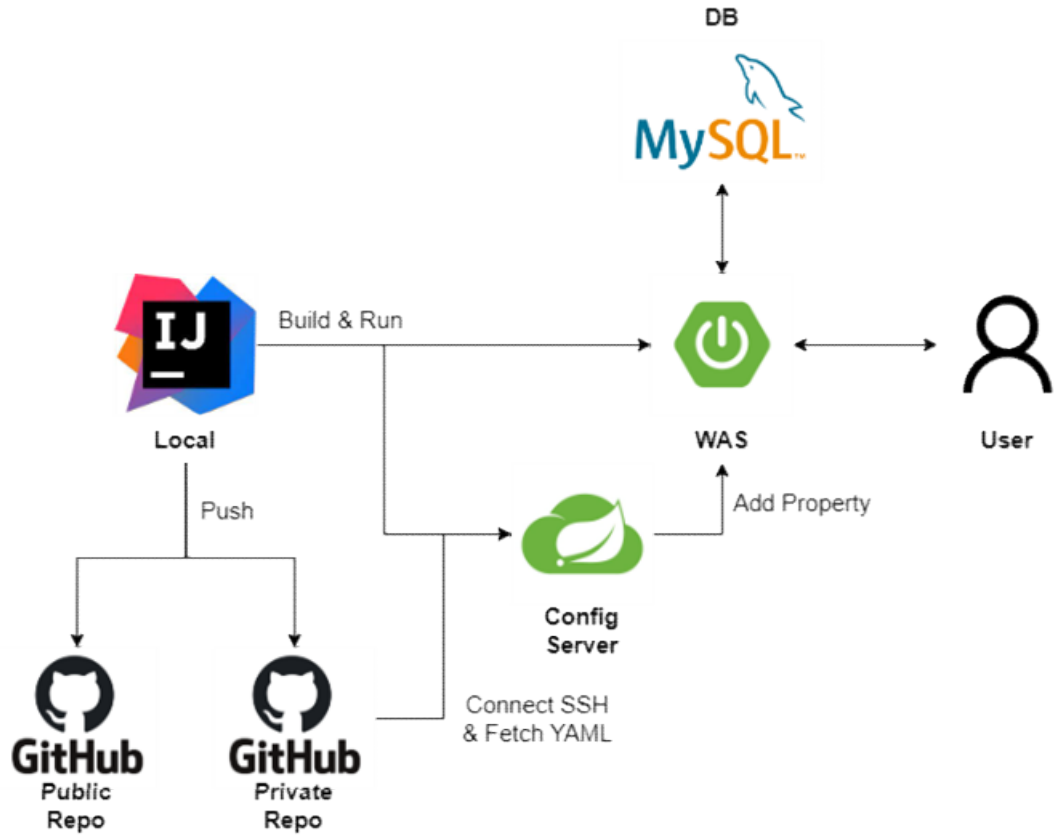
[Figure. 1] Project Initial Configuration

[Development environment]
IDE : IntelliJ IDEA 2021.03
WAS : Spring Boot 2.7.0
DB : MySQL 8.0
Build : Gradle 7.4

[Feature]
- User request directly to WAS via browser.
- WAS has **MVC Pattern.** (Use 'Thymeleaf' as Template Engine)
- Authentication, API and HTML/CSS/JS management in the WAS. **(Monolithic)**
- Develops with attention to DDD(Domain Driven Design) and TDD(Test Driven Development).

- In this step, 'Github' is at the level for code backup.

1. First Step



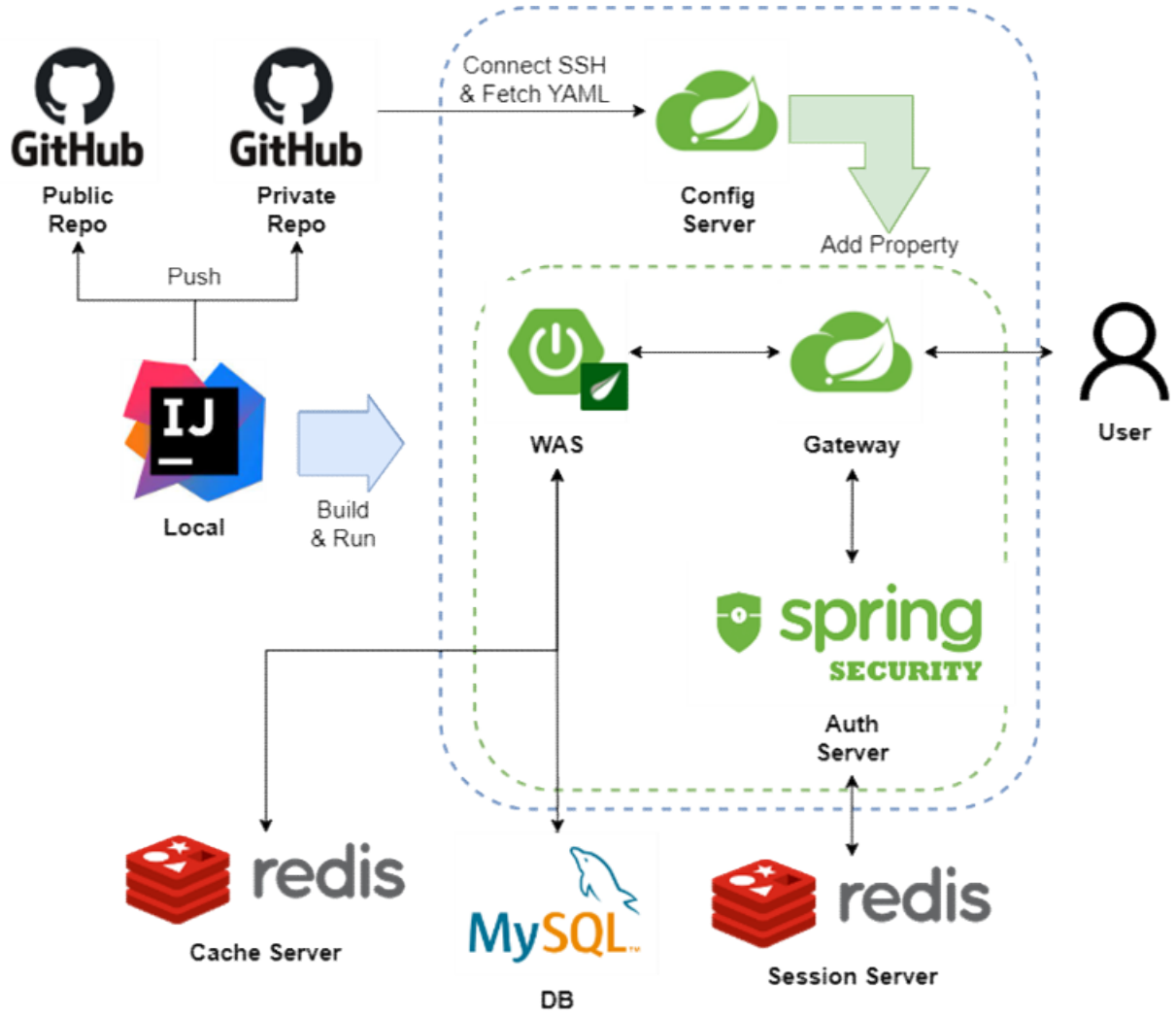
[Figure. 2] Add Config Server

[Add]
Config Server : Spring Cloud Config Server 3.1.4

[Feature]
- Config Server enables **efficient management of configuration files(YAML)** when expanding additional services.
- **Credential is encrypted.** (Enhance security by setting the Repository to Private)
- When Running Config Server, Connect to Github Repo via SSH.
- When Running WAS, Add properties after request to Config Server.
- Therefore, the **Config Server must be turned on first.**

- Establish a Git Branch management strategy : Adopt **Github Flow**

2. Second Step

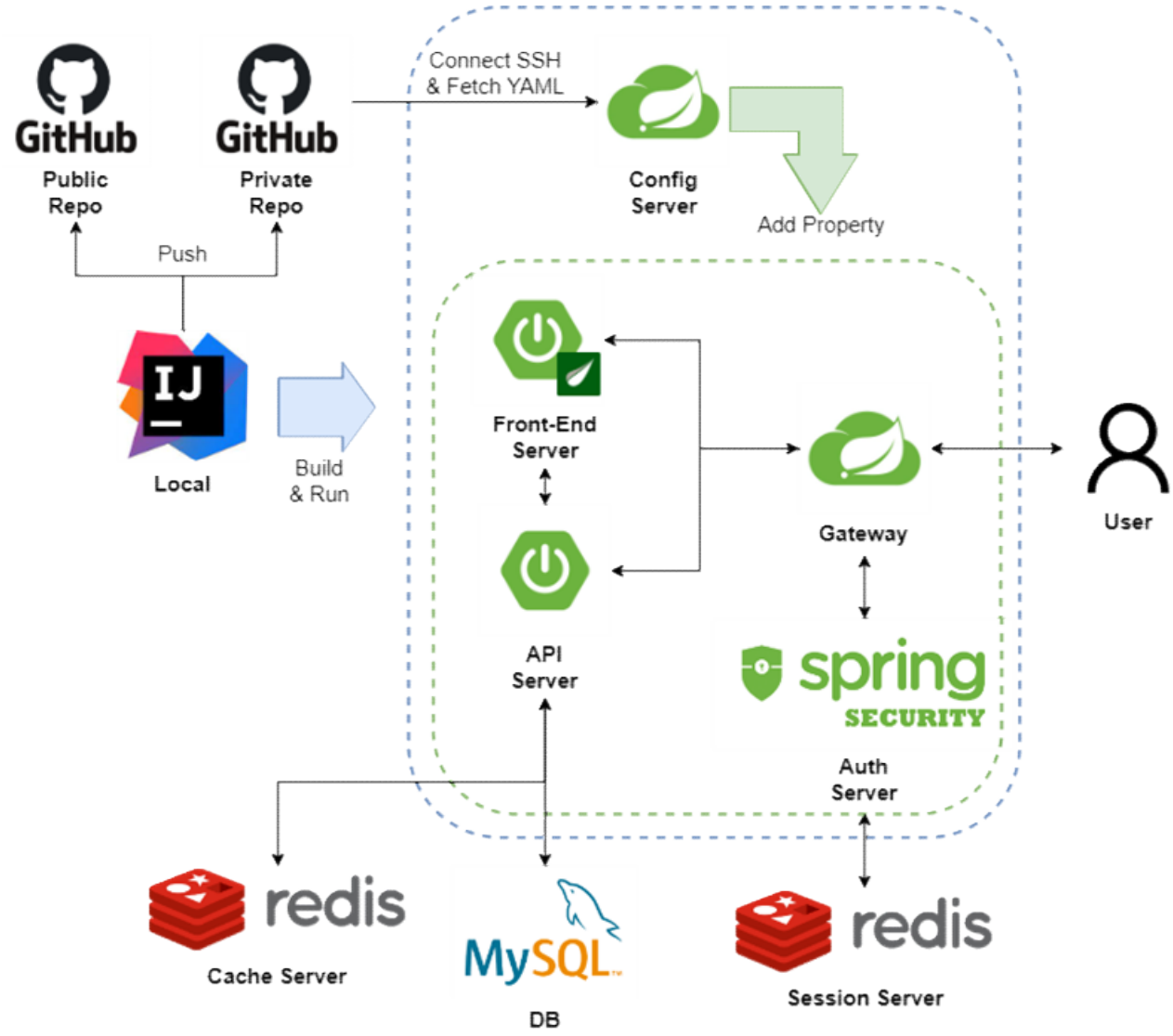


[Figure. 3] Detach Authentication Server and Add Cache Server

[Add]
Gateway : Spring Cloud Gateway
Cache/Session Server : Redis

[Feature]
- **Separated authentication** features of existing WAS **to independent server.**
- **Add Gateway** to facilitate service expansion.
- **JWT authentication and routing** to WAS upon request by the user to WAS.
- **Add Session Server** to store JWT.
  (Persistent DB should also be configured for actual operation, but pass due to lack of resources)
- **Add Cache Server** for DB Load Balancing and Faster Response.

3. Third Step

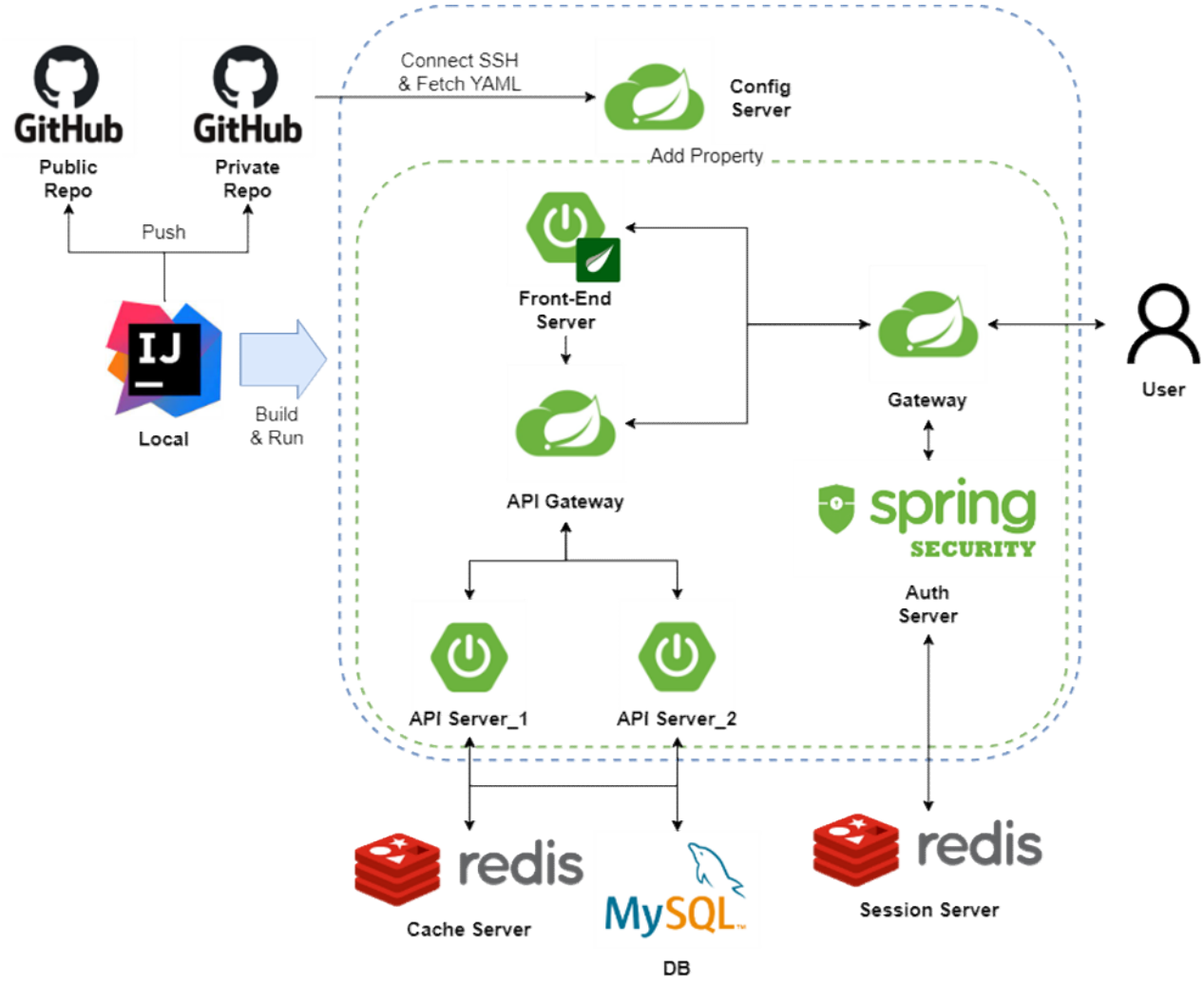


[Figure. 4] Separates Front-End Server and API Server

[Feature]

- **Separates view and business logic** from WAS
- Front-End Server can get data by requesting API Server if there is any necessary.
- The separated Front-End server and API server are connected to the Gateway.
- Users can use the web through Front-End or receive data only through API server.

4. Fourth Step

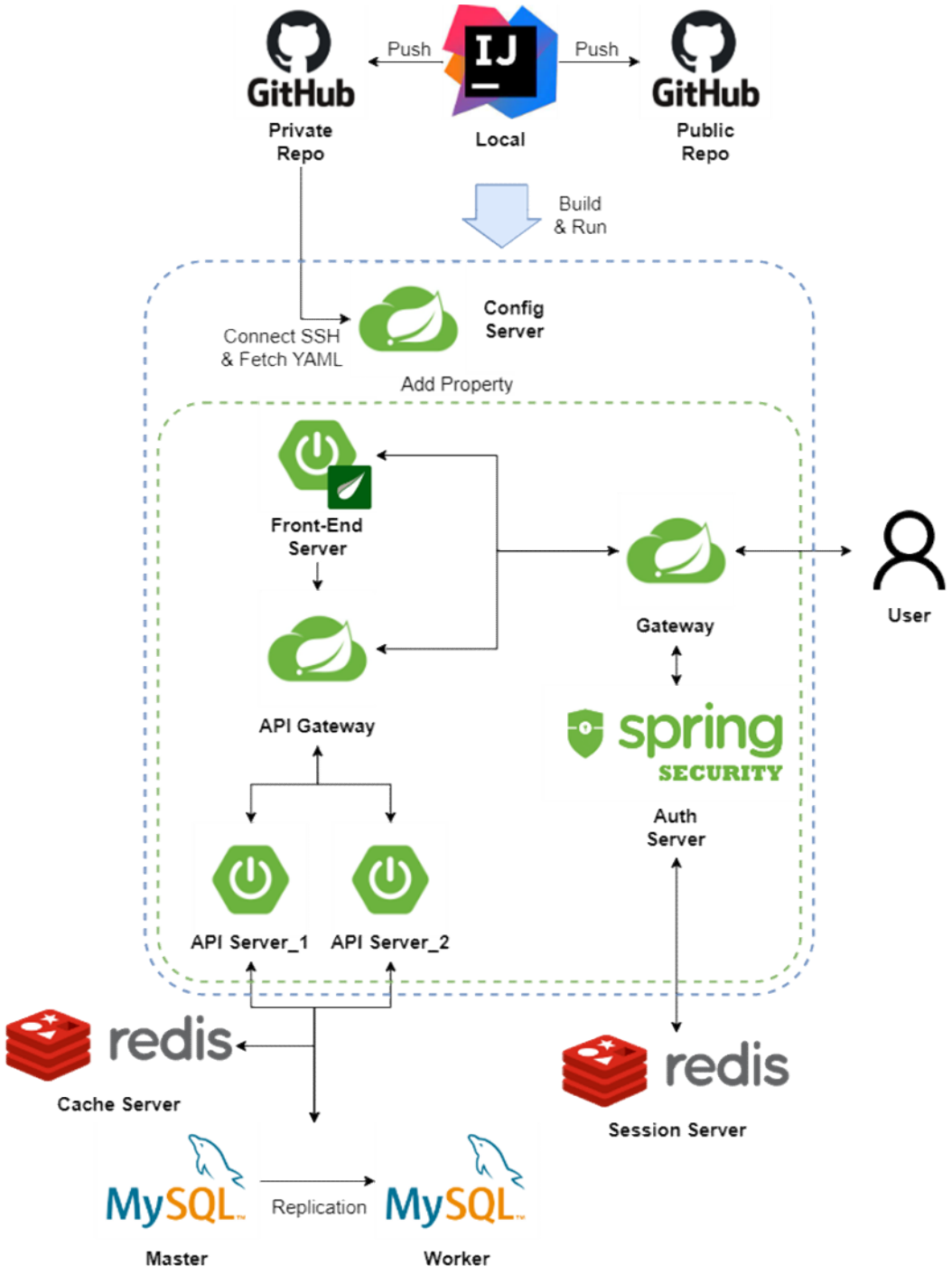


[Figure. 5] API Server Redundancy and Add API Gateway

[Feature]

- **Server Redundancy** for load balancing and fast processing
- **Add API Gateway** for Load Balancing (strategy : **Round Robin**)
- API Server are growing, but **DB is experiencing performance issues** because they share one database

5. Fifth Step

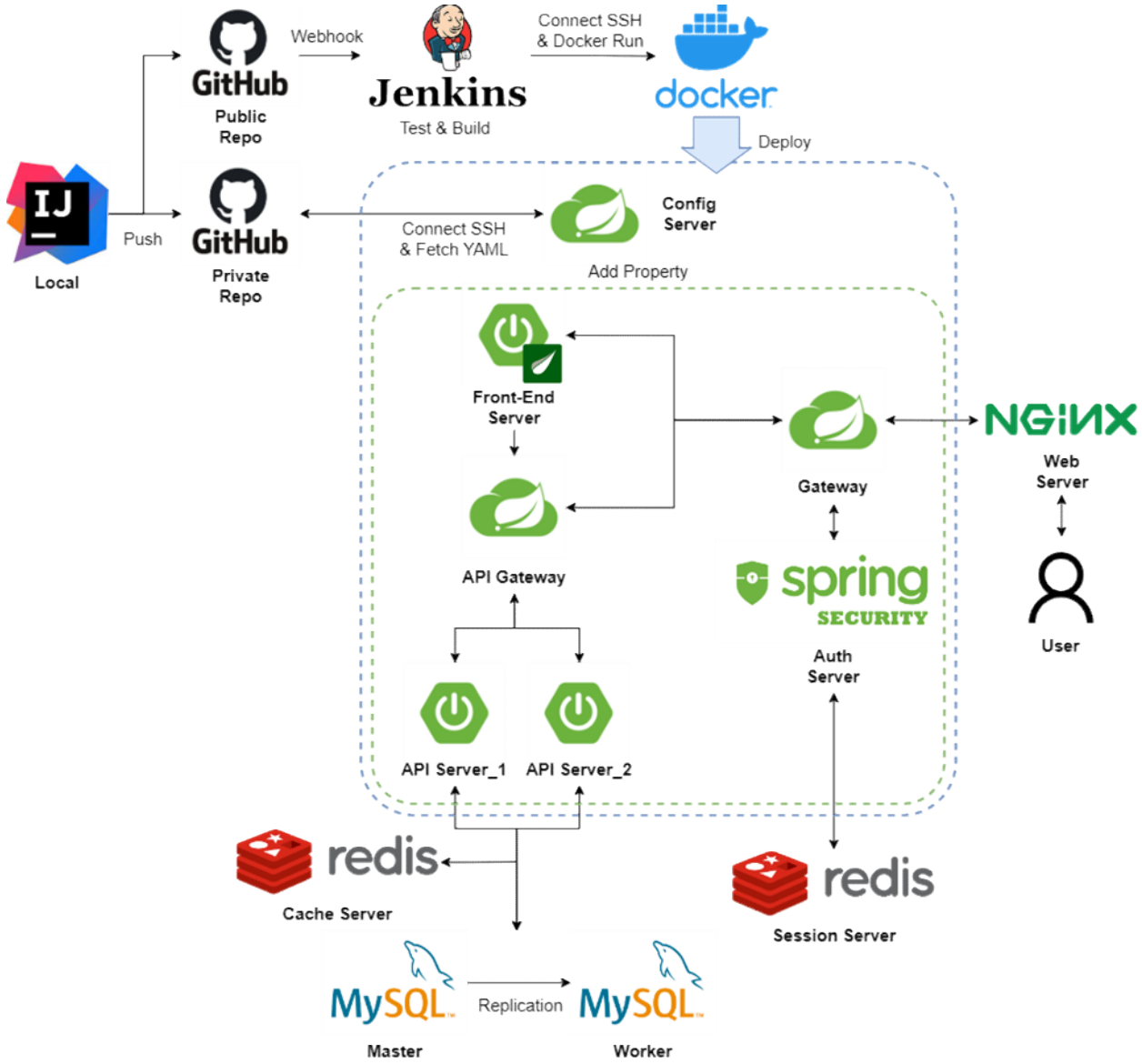


[Figure. 6] Database Redundancy

[Feature]

- Perform '**MySQL' Replication** to accommodate increased API server.
- Write/Read on Master node but only Read on Worker node.
- The process from the front-end server to the DB has increased, so a **caching strategy needs to be established after performance testing.**

6. Final Step



[Figure. 7] Configuring CI/CD Pipelines and Add Web Server

[Add]
CI/CD : Jenkins
Container : Docker

[Feature]
- **Automate** the traditional manual **build and deployment process** by organizing CI/CD pipelines with 'Jenkins' and 'Docker'.
- 'Jenkins' tests and builds after "Webhook" in Public Repository (CI)
- **Push Archive Artifacts** to Docker Hub (CD)
- **Connecting Docker to SSH and Deploy** (CD)

- **Add 'Nginx'** to Web Server to Manage Static Files Separately
- **Route to the gateway** when URL requests requiring authentication are received